UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/721,879	11/26/2003	Harold Theodore Devor	P-6216-US	6009

49444 7590 05/13/2008

PEARL COHEN ZEDEK LATZER, LLP
1500 BROADWAY, 12TH FLOOR
NEW YORK, NY 10036

EXAMINER	
FENNEMA, ROBERT E	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
05/13/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/721,879	DEVOR ET AL.	
	Examiner	Art Unit	
	ROBERT E. FENNEMA	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *28 January 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1, 3-9, 11-17, 19-21, and 23-32* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3-9,11-17,19-21 and 23-32* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1, 3-9, 11-17, 19-21, and 23-32 are pending. Claims 1, 9, 17, 21, and 24 amended as per Applicant's request. Claims 29-32 added as per Applicants request.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1, 3-7, 9, 11-15, 21, 23-26, 28-30 and 32 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hohensee et al. (USPN 6,064,815, herein Hohensee), in view of Angel et al. (USPN 6,643,842, herein Angel).

4. As per Claim 1, Hohensee teaches: A method comprising:

during translation of a code block from a first format suitable for a first computing platform to a second format suitable for a second computing platform (Column 1, Lines 45-47 and Column 1, Line 62-Column 2, Line 4), to detect whether execution of said code block results in the misaligned data access prior to execution of said code block (Column 3, Lines 4-9 shows the detector detecting an exceptional condition, and Column 2, Lines 61-67 show the exception to be caused by a misaligned memory reference);

storing a location of a first memory address if accessing the first memory address by a first instruction results in the misaligned data access (Column 3, Lines 9-15, in order to start the fix-up code generator, some storing of the address must be done to pass it along);

adding one or more instructions to the first instruction (Column 3, Line 15, the fix-up code);

checking if a location of a second memory address is identical to the location of the first memory address when a second instruction requires access to the second memory address (Column 3, Lines 29-35, the second instruction is the first instruction being encountered again in a loop); and

obviating the need to add instructions to the second instruction if the location of the second memory address is accessed by the second instruction is identical to the location of the first memory address (Column 3, Lines 29-35, when the instruction is encountered again, there is no need to add fix-up code again, as the problem has already been solved once), but fails to teach:

inserting one or more instructions in said code block.

While Hohensee teaches that there is a detection of misaligned data access prior to the execution of a code block, and a correction of said misalignment by adding instructions, Hohensee teaches that it is done through an exception condition detector, which is not given much more detail, and there is no indication in Hohensee that the exception condition detector is instructions inserted into the code block. Therefore, while

Hohensee teaches the same end result as the claim, the way in which it is detected is presumably different. However, Angel teaches that one of the most prevalent run-time errors are relating to memory access, and that the prior art way of dealing with these issues was to insert code to monitor the performance of the code, and to provide indications when an improper access occurs (Column 1, Lines 26-37). Given that Angel teaches that this is the method used to detect memory problems (the problem also posed by Honhensee), and that it is a prior art method of detecting the problem (the time frame of the invention is the time frame which Honhensee was invented), one of ordinary skill in the art, at the time the invention was made, lacking a specific way to detect the misaligned data accesses which Honhensee detects, and with Angel describing a way to do so, would have implemented Angel's method of inserting code into the software in order to implement the "detection" as described by Honhensee.

5. As per Claim 3, Hohensee teaches: The method of claim 1, wherein detecting comprises inserting at least one instruction in said code block to detect a location of an instruction whose execution results in the misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

6. As per Claim 4, Hohensee teaches: The method of claim 1, wherein inserting one or more instructions in of said code block comprises inserting at least one instruction in said code block to detect a location of an instruction whose execution results in the

misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

7. As per Claim 5, Hohensee teaches: The method of claim 1, wherein modifying comprises adding to said code block an instruction to branch an execution of said code block to a code sequence whose execution handles the misaligned data access (Column 12, Lines 23-30. A branch to a Fixup code block is called to handle the misalignment).

8. As per Claim 6, Hohensee teaches: The method of claim 1, wherein modifying comprises modifying said code block to handle misaligned data access in a subsequent execution of said code block (Column 12, Lines 23-30. A branch to a Fixup code block is called to handle the misalignment).

9. As per Claim 7, Hohensee teaches: The method of claim 1, further comprising translating said code block from said first format to said second format (Column 1, Line 62 – Column 2, Line 4).

10. As per Claim 9, Hohensee teaches: An apparatus comprising:

a processor, during translation of a code block from a first format suitable for a first computing platform to a second format suitable for a second computing platform (Column 1, Lines 45-47 and Column 1, Line 62-Column 2, Line 4), in said code block to

detect whether execution of said code block results in misaligned data access prior to execution (Column 3, Lines 4-9 shows the detector detecting an exceptional condition, and Column 2, Lines 61-67 show the exception to be caused by a misaligned memory reference), of said code block,

to store a location of a first memory address if accessing the first memory address by a first instruction results in the misaligned data access (Column 3, Lines 9-15, in order to start the fix-up code generator, some storing of the address must be done to pass it along);

to add one or more instructions to the first instruction (Column 3, Line 15, the fix-up code) to check if a location of a second memory address is identical to the location of the first memory address when a second instruction requires access to the second memory address (Column 3, Lines 29-35, the second instruction is the first instruction being encountered again in a loop); and

to obviate the need to add instructions to the second instruction if the location of the second memory address is accessed by the second instruction is identical to the location of the first memory address (Column 3, Lines 29-35, when the instruction is encountered again, there is no need to add fix-up code again, as the problem has already been solved once), but fails to teach:

inserting one or more instructions during translation.

While Hohensee teaches that there is a detection of misaligned data access prior to the execution of a code block, and a correction of said misalignment, Hohensee teaches that it is done through an exception condition detector, which is not given much

more detail, and there is no indication in Hohensee that the exception condition detector is instructions inserted into the code block. Therefore, while Hohensee teaches the same end result as the claim, the way in which it is detected is presumably different. However, Angel teaches that one of the most prevalent run-time errors are relating to memory access, and that the prior art way of dealing with these issues was to insert code to monitor the performance of the code, and to provide indications when an improper access occurs (Column 1, Lines 26-37). Given that Angel teaches that this is the method used to detect memory problems (the problem also posed by Honhensee), and that it is a prior art method of detecting the problem (the time frame of the invention is the time frame which Honhensee was invented), one of ordinary skill in the art, at the time the invention was made, lacking a specific way to detect the misaligned data accesses which Honhensee detects, and with Angel describing a way to do so, would have implemented Angel's method of inserting code into the software in order to implement the "detection" as described by Honhensee.

11. As per Claim 11, Hohensee teaches: The apparatus of claim 9, wherein the processor is able to insert at least one instruction in said code block to detect a location of an instruction whose execution results in the misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

12. As per Claim 12, Hohensee teaches: The apparatus of claim 9, wherein the processor is able to insert at least one instruction in said code block to detect a location of an instruction whose execution results in the misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

13. As per Claim 13, Hohensee teaches: The apparatus of claim 9, wherein the processor is able to add to said code block an instruction to branch an execution of said code block to a code sequence whose execution handles the misaligned data access (Column 12, Lines 23-30. A branch to a Fixup code block is called to handle the misalignment).

14. As per Claim 14, Hohensee teaches: The apparatus of claim 9, wherein the processor is able to modify said code block to handle misaligned data access in a subsequent execution of said code block (Column 12, Lines 23-30. A branch to a Fixup code block is called to handle the misalignment).

15. As per Claim 15, Hohensee teaches: The apparatus of claim 9, wherein the processor is able to, before insertion, translate said code block from said first format to said second format (Column 1, Line 62 – Column 2, Line 4).

16. As per Claim 21, Hohensee teaches: A machine-readable medium having stored thereon a set of instructions that, if executed by a machine, cause the machine to perform a method comprising:

during translation of a code block from a first format suitable for a first computing platform to a second format suitable for a second computing platform (Column 1, Lines 45-47 and Column 1, Line 62-Column 2, Line 4) detecting whether execution of said code block results in the misaligned data access prior to execution of said code block (Column 3, Lines 4-9 shows the detector detecting an exceptional condition, and Column 2, Lines 61-67 show the exception to be caused by a misaligned memory reference); and

storing a location of a first memory address if accessing the first memory address by a first instruction results in the misaligned data access (Column 3, Lines 9-15, in order to start the fix-up code generator, some storing of the address must be done to pass it along);

adding one or more instructions to the first instruction (Column 3, Line 15, the fix-up code);

checking if a location of a second memory address is identical to the location of the first memory address when a second instruction requires access to the second memory address (Column 3, Lines 29-35, the second instruction is the first instruction being encountered again in a loop); and

obviating the need to add instructions to the second instruction if the location of the second memory address accessed by the second instruction is identical to the

location of the first memory address (Column 3, Lines 29-35, when the instruction is encountered again, there is no need to add fix-up code again, as the problem has already been solved once), but fails to teach:

inserting one or more instructions in said code block.

While Hohensee teaches that there is a detection of misaligned data access prior to the execution of a code block, and a correction of said misalignment, Hohensee teaches that it is done through an exception condition detector, which is not given much more detail, and there is no indication in Hohensee that the exception condition detector is instructions inserted into the code block. Therefore, while Hohensee teaches the same end result as the claim, the way in which it is detected is presumably different. However, Angel teaches that one of the most prevalent run-time errors are relating to memory access, and that the prior art way of dealing with these issues was to insert code to monitor the performance of the code, and to provide indications when an improper access occurs (Column 1, Lines 26-37). Given that Angel teaches that this is the method used to detect memory problems (the problem also posed by Honhensee), and that it is a prior art method of detecting the problem (the time frame of the invention is the time frame which Honhensee was invented), one of ordinary skill in the art, at the time the invention was made, lacking a specific way to detect the misaligned data accesses which Honhensee detects, and with Angel describing a way to do so, would have implemented Angel's method of inserting code into the software in order to implement the "detection" as described by Honhensee.

17. As per Claim 23, Hohensee teaches: The machine-readable medium of claim 21, wherein the instructions that result in detecting result in insertion of at least one instruction in said code block to detect a location of an instruction whose execution results in the misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

18. As per Claim 24, Hohensee teaches: The machine-readable medium of claim 21, wherein the instructions that result in insertion result in insertion of at least one instruction in said code block to detect a location of an instruction whose execution results in the misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

19. As per Claim 25, Hohensee teaches: The machine-readable medium of claim 21, wherein the instructions comprise at least part of a translator (Column 1, Lines 45-48 disclose a translator).

20. As per Claim 26, Hohensee teaches: The machine-readable medium of claim 21, wherein the instructions comprise at least part of an execution layer (The instructions run throughout Hohensee's invention are executed, which necessitate them being in the execution layer).

21. As per Claim 28, Hohensee teaches: The machine-readable medium of claim 21, wherein the instructions comprise at least part of a compiler (Column 1, Lines 45-48, where a translator is a compiler).

1. As per Claim 29, Hohensee teaches: The method of claim 1, wherein the location of the second memory address is identical to the location of the first memory address if the size of the difference between the first and second memory addresses is the same as the size of the data misalignment or is a factor of the size of the data misalignment (Column 3, Lines 29-35, it's the same in this case).

30. The apparatus of claim 9, wherein the location of the second memory address is identical to the location of the first memory address if the size of the difference between the first and second memory addresses is the same as the size of the data misalignment or is a factor of the size of the data misalignment (Column 3, Lines 29-35, it's the same in this case).

32. The machine-readable medium of claim 21, wherein the location of the second memory address is identical to the location of the first memory address if the size of the difference between the first and second memory addresses is the same as the size of the data misalignment or is a factor of the size of the data misalignment (Column 3, Lines 29-35, it's the same in this case).

22. Claims 8, 16-17, 19-20, and 31 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hohensee and Angel.

23. As per Claim 8, Hohensee teaches the method of claim 1, but fails to teach:

wherein inserting one or more instructions in said code block further comprises inserting one or more instructions in said code block to detect whether execution of said code block results in the a misaligned data access prior to execution of a code block translated from a format suitable for a 32-bit based computing platform to a format suitable for a 64-bit based computing platform.

Hohensee teaches that a host processor, in an execution environment, may emulate operations performed by an emulated microprocessor, but not th sizes of the processor and the emulated processor. The Examiner is taking official notice that it is well known in the art that most computer processors operate on a number of bits that are a power of 2, for example, 8, 16, 32, 64, and 128, and that a primary difference between computer processors (of the same or similar instruction set) are a difference in the bit-size of the processors. *In re Rose*, 220 F.2d 459, 463, 105 USPQ 237, 240 (CCPA 1955) teaches that it is within the skill of one of ordinary skill in the art to change size, so whether the emulation required was from 8 to 16 bits, 16 to 32 bits, or 32 to 64 bits is irrelevant to one of ordinary skill in the art. Therefore, one of ordinary skill in the art would have been able to make use of Hohensees invention, and apply it to a 64-bit processor running a 32-bit program.

24. As per Claim 16, Hohensee teaches the apparatus of claim 9, but fails to teach: wherein the first computing platform is a 32-bit based computing platform and the second computer architecture is a 64-bit based computing platform. Hohensee teaches that a host processor, in an execution environment, may emulate operations performed by an emulated microprocessor, but not the sizes of the processor and the emulated processor. The Examiner is taking official notice that it is well known in the art that most computer processors operate on a number of bits that are a power of 2, for example, 8, 16, 32, 64, and 128, and that a primary difference between computer processors (of the same or similar instruction set) are a difference in the bit-size of the processors. *In re Rose*, 220 F.2d 459, 463, 105 USPQ 237, 240 (CCPA 1955) teaches that it is within the skill of one of ordinary skill in the art to change size, so whether the emulation required was from 8 to 16 bits, 16 to 32 bits, or 32 to 64 bits is irrelevant to one of ordinary skill in the art. Therefore, one of ordinary skill in the art would have been able to make use of Hohensee's invention, and apply it to a 64-bit processor running a 32-bit program.

25. As per Claim 17, Hohensee teaches: A computing platform comprising:

a processor, during translation of a code block from a first format suitable for a first computing platform to a second format suitable for a second computing platform (Column 1, Lines 45-47 and Column 1, Line 62-Column 2, Line 4), in said code block to detect whether execution of said code block results in misaligned data access prior to execution (Column 3, Lines 4-9 shows the detector detecting an exceptional condition,

and Column 2, Lines 61-67 show the exception to be caused by a misaligned memory reference) of said code block,

to store a location of a first memory address if accessing the first memory address by a first instruction results in the misaligned data access (Column 3, Lines 9-15, in order to start the fix-up code generator, some storing of the address must be done to pass it along);

to add one or more instructions to the first instruction (Column 3, Line 15, the fix-up code);

to check if a location of a second memory address is identical to the location of the first memory address when a second instruction requires access to the second memory address (Column 3, Lines 29-35, the second instruction is the first instruction being encountered again in a loop); and

to obviate the need to add instructions to the second instruction if the location of the second memory address accessed by the second instruction is identical to the location of the first memory address (Column 3, Lines 29-35, when the instruction is encountered again, there is no need to add fix-up code again, as the problem has already been solved once); and

a dynamic random access memory operably associated with said processor to store at least a portion of said code block (Figure 1 discloses a memory, but Hohensee does not explicitly teach the memory being a dynamic random access memory (herein DRAM). However, the Examiner is taking official notice that using a DRAM for computer memory is well known in the art, due to its cheap cost and widespread use, which would

have motivated one of ordinary skill in the art to utilize a DRAM in Hohensee's invention), but fails to teach:

inserting one or more instructions.

While Hohensee teaches that there is a detection of misaligned data access prior to the execution of a code block, and a correction of said misalignment, Hohensee teaches that it is done through an exception condition detector, which is not given much more detail, and there is no indication in Hohensee that the exception condition detector is instructions inserted into the code block. Therefore, while Hohensee teaches the same end result as the claim, the way in which it is detected is presumably different. However, Angel teaches that one of the most prevalent run-time errors are relating to memory access, and that the prior art way of dealing with these issues was to insert code to monitor the performance of the code, and to provide indications when an improper access occurs (Column 1, Lines 26-37). Given that Angel teaches that this is the method used to detect memory problems (the problem also posed by Honhensee), and that it is a prior art method of detecting the problem (the time frame of the invention is the time frame which Honhensee was invented), one of ordinary skill in the art, at the time the invention was made, lacking a specific way to detect the misaligned data accesses which Honhensee detects, and with Angel describing a way to do so, would have implemented Angel's method of inserting code into the software in order to implement the "detection" as described by Honhensee.

26. As per Claim 19, Hohensee teaches: The apparatus of claim 17, wherein the processor is able to insert at least one instruction in said code block to detect a location of an instruction whose execution results in the misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

27. As per Claim 20, Hohensee teaches: The apparatus of claim 17, wherein the processor is able to insert at least one instruction in said code block to detect a location of an instruction whose execution results in the misaligned data access (Column 3, Lines 15-17. In order to substitute code for an instruction, the location would necessarily have to have been detected).

2. As per Claim 31, Hohensee teaches: The apparatus of claim 17, wherein the location of the second memory address is identical to the location of the first memory address if the size of the difference between the first and second memory addresses is the same as the size of the data misalignment or is a factor of the size of the data misalignment (Column 3, Lines 29-35, it's the same in this case).

28. Claim 27 is rejected under 35 U.S.C. 103(a) as being unpatentable over Hohensee, in view of Drongowski.

29. As per Claim 27, Hohensee teaches: The machine-readable medium of claim 21, wherein the instructions comprise at least part of an operating system. While Hohensee does not explicitly disclose an operating system, it would have been very obvious to one of ordinary skill in the art to be able to make use of misalignment correction capabilities on the operating-system level, so that all programs and programmers can make use of it, as well as the fact that operating systems are extremely common on most computing systems. Drongowski teaches an example of an operating system (The Alpha Linux) that makes use of commands to fix alignment problems (Section 2.7), and explains the problems misalignment can cause. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to allow an operating system to run these instructions and make use of Hohensee's invention.

Response to Arguments

3. Applicant has argued that Hohensee and Angel do not teach the currently amended independent claims, which claim (paraphrasing) storing a location of an address if an instruction is going to result in a misaligned data access, adding instructions to the instruction, checking to see if a second instruction accesses a memory address which is identical to the first memory address, and not adding instructions if they are identical. Examiner believes Hohensee teaches these limitations, as explained in the claim rejections, and explained again here: Hohensee teaches that when the exceptional condition detector detects a misaligned address, a subsystem is called to insert fix up code before the instruction to prevent said misaligned address.

The only way this could be done is to store the address somewhere, so that the subsystem has access to the memory location to fix. Without knowing more details about where or how the address is stored, Hohensee reads on this limitation. Further, the inserting of the fix-up code reads on the adding of instructions. Regarding the limitations of checking if a location of a second memory address is identical to the first in a second instruction, and not adding instructions if that is the case, Hohensee discusses in Column 3 that once the problem is fixed, there is no need to fix it anymore. For example, if the instruction is part of a loop, once it has been fixed once, there will never be another need to fix (Add instructions) to it again. And obviously, an instruction in a loop being referred to multiple times would continue to have the same memory address.

4. In order to potentially overcome this rejection, the Examiner can suggest a few avenues for the Applicant to consider. The most apparent to the Examiner would be to further elaborate on the storing of the address, if there is anything that is done specifically, it would greatly help to distinguish from the prior art. Other options may be to further clarify the added instructions, or the first and second instructions themselves.

Conclusion

5. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within

TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to ROBERT E. FENNEMA whose telephone number is (571)272-2748. The examiner can normally be reached on Monday-Friday, 8:30-6:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Eddie P Chan/ Robert E Fennema

Supervisory Patent Examiner, Art Unit 2183

Examiner
Art Unit 2183

RF